A column generation-based heuristic for rostering with work patterns

Lusby, Richard Martin; Dohn, Anders Høeg; Range, Troels Martin; Larsen, Jesper

Published in:
Journal of the Operational Research Society

Link to article, DOI:
10.1057/jors.2011.27

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Journal of the Operational Research Society (2011) 1–17

www.palgrave-journals.com/jors/

A column generation-based heuristic for rostering with work patterns

R Lusby[1*], A Dohn[1], TM Range[2] and J Larsen[1]

[1]*Technical University of Denmark, Lyngby, Denmark; and* [2]*University of Southern Denmark, Odense M, Denmark*

This paper addresses the Ground Crew Rostering Problem with Work Patterns, an important manpower planning problem arising in the ground operations of airline companies. We present a cutting stock-based integer programming formulation of the problem and describe a powerful heuristic decomposition approach, which utilizes column generation and variable fixing, to construct efficient rosters for a six-month time horizon. The time horizon is divided into smaller blocks, where overlaps between the blocks ensure continuity. The proposed methodology is able to circumvent one step of the conventional roster construction process by generating rosters directly based on the estimated workload. We demonstrate that this approach has the additional advantage of being able to easily incorporate robustness in the roster. Computational results on real-life instances confirm the efficiency of the approach.

Journal of the Operational Research Society advance online publication, 4 May 2011
doi:10.1057/jors.2011.27

Keywords: manpower planning; optimization; cutting stock problem; column generation

1. Introduction

In this paper, we consider the Ground Crew Rostering Problem with Work Patterns (GCRPWP) for a major European airline. Ground crew comprises all of the crew that an airline employs at an airport to take care of passengers and aircrafts in order to facilitate a smooth operation. This could be, for instance, customer service representatives or ramp service workers. The rostering of these workers is a complex, multi-stage planning process, which starts with the initial forecast of labour requirements and concludes with the construction of a roster that covers the anticipated workload as well as possible. The roster specifies what days of a pre-specified time horizon each employee will work as well as the type of work they will do. At this airline, the roster is published before the start of each season (summer and winter) and states what each person will be doing for the next six months.

Rostering staff can be seen as the process of assigning an employee to a sequence of shifts. A shift refers to a block of work, typically around nine hours in duration and is associated with a specific task. Here, it is assumed that shifts have already been defined and the challenge is to generate an efficient roster for the employees while respecting the legislation and the staff agreements. The most important requirement, in this particular problem, is that all staff work the same *work pattern*. A work pattern specifies the number of consecutive days of work as well as the number of required days of rest in between. For instance, this airline uses a 6&3 work pattern, which states that an employee will be assigned six days of work and then receive three days rest. Of course, the work pattern must be staggered across the employees to ensure that they are not all off on the same days. Hence, associated with any work pattern is a set of *day-off patterns*, the cardinality of which is equal to the number of days in the work pattern. The obtained roster should typically cover the workload, while ensuring a certain degree of robustness. Uncovered work is allowed, but incurs a penalty. Robustness is incorporated by providing over coverage on the estimated workload.

In this paper, we propose a cutting stock-based mixed integer programming (MIP) formulation of the problem. Initially, it is assumed that the required staffing level is specified for each shift. To solve the model, we decompose the six-month time horizon into smaller, computationally tractable blocks. A procedure that combines column generation and variable fixing is developed to solve each block. The blocks are solved sequentially and consistency between the rosters of successive blocks is enforced by fixing shifts in the overlaps of blocks. The initial model is then extended to generate rosters directly on the forecast workload, and shift demands are made dispensable. The number of employees working any given shift is determined

*Correspondence: R Lusby, Department of Management Engineering, Technical University of Denmark, Produktionstorvet, Building 426, 2800 Kgs. Lyngby, Denmark.

by the new model as a part of the solution and robustness is built into the solution. Finally, we test and compare the proposed methodology on several instances arising in practice.

The outline of the paper is as follows. Section 2 presents a review of the research in this area. Section 3 provides a more formal definition of the problem and presents the mathematical programming formulation. A discussion of the proposed optimization-based heuristic is given in Sections 4 and 5. In Section 6, the initial model is modified to generate rosters directly from the workload, while Section 7 explains how we incorporate robustness. Computational results on real-life instances are presented in Section 8 and conclusions from this research are summarized in Section 9.

2. Literature review

Crew rostering is a classical optimization problem. It is a very important problem as people are often both a critical and an expensive part of the operations. Utilizing the available manpower as effectively as possible can lead to significant potential savings. Furthermore, good crew planning ensures a high job satisfaction, which in turn results in higher productivity.

An improved productivity can be obtained by using computerized decision-support tools based on advanced optimization techniques. The development has been pioneered by the airline industry (see eg Barnhart *et al*, 2003). Nowadays, the rostering of pilots and cabin crew without such tools is unthinkable for all of the major airlines. In Butchers *et al* (2001), the authors estimate the annual savings of Air New Zealand to be around NZD 15 million, more than 6% of the annual estimated crew costs. Similarly, in Anbil *et al* (1991) annual savings in excess of USD 20 million are reported. This is roughly 1% of the annual estimated crew costs. The underlying optimization techniques have since penetrated into other areas of manpower planning and rostering. For example, many train companies now use optimization methods for rostering drivers and conductors (see eg Abbink *et al*, 2005). Other prominent areas of rostering are call centres, protection and emergency services, venue management, retail, and civic services. A review can be found in Ernst *et al* (2004).

While there has been a large and continued focus on optimization within the rostering of pilots and cabin crew, the successful results have not lead airlines to thoroughly investigate the potential of applying similar methods to the rostering of ground crew despite the fact they are, from a modelling perspective, very similar problems. Ground crew rostering does, however, not have the added restriction that a crew member may end up in a different location to that where his/her roster started nor the disparity in qualification levels of its crew members. The rostering of cabin crew and pilots is also heavily constrained by strict union rules as well as civil aviation rules. One noticeable aspect in which ground crew rostering is more difficult is in the length of the rosters that must be generated (typically 6 months); this makes it difficult to apply standard mathematical programming techniques.

One of the few papers that addresses ground crew rostering is by Dowling *et al* (1997). The authors present a solution approach for rostering around 500 staff at a large international airport. The proposed algorithm is based on simulated annealing and rosters airline ground staff over a monthly planning period, where the objective is to minimize idle time. Other contributions include Brusco *et al* (1995) and Chu (2007). Brusco *et al* describe a manpower planning tool for United Airlines (UA). This tool produces tour schedules for which employees bid using a seniority-based system and is used by UA at over 100 airports. It is a two-phase approach; the first phase generates requirements for labour using a set cover formulation, while the second phase is a simulated annealing-based metaheuristic that attempts to improve the tours found in the first phase. Chu proposes a goal programming approach to generate daily schedules for baggage handling at Hong Kong International Airport.

Despite not being a well-studied problem itself, the GCRPWP does bare strong similarities to many other rostering problems arising in various contexts. In particular, nurse rostering and physician scheduling, both of which arise in the area of health care, are two problems which possess the strongest similarities. In nurse rostering, one must usually provide suitably qualified nurses to cover the workload demand based on the number of patients in the ward, while satisfying a wide range of local and national working regulations. Similarly, in physician scheduling, one must construct rosters for doctors at hospitals so each shift of every day is covered by exactly one physician. Although these problems can be modelled similarly, they are slightly more complicated than the GCRPWP. Firstly, the lengths of the on and off periods are typically not fixed, as is the case here, but should be within certain bounds. Furthermore, both problems attempt to satisfy as many individual staff requests as possible and thus must generate sequences of shifts specific to each individual nurse/physician. Given the fixed nature of a work pattern, it is impossible in this context to take into account individual preferences such as particular weekends on or off. Furthermore, in the GCRPWP, all staff are assumed to be equally qualified in that they can work any shift. For these reasons, we attempt to construct anonymous sequences of shifts for groups of staff members. Nurse rostering and physician scheduling models can easily be adapted for the ground crew rostering problem; however, as is the case with the rostering of cabin crew and pilots, the time horizon is much shorter. This makes it difficult to apply their solution methodologies directly. The most

recent surveys on the nurse rostering problem are Cheang *et al* (2003) and Burke *et al* (2004), while a good overview on physician scheduling can be found in Gendreau *et al* (2006). In addition, some interesting questions on the lack of transition from academia to industry are raised in Kellogg and Walczak (2007).

The use of work patterns is, however, also not uncommon in rostering problems. For example, Alfares (2002) describes a particular rostering problem where a 14&7 work pattern is required. The author considers cyclic weekly demand and, besides minimizing the size of the labour force, attempts to use as few different day-off patterns as possible. Both Vohra (1987) and Alfares (1997) consider day off scheduling assuming a 5&2 work pattern. Alfares (1997) presents a two-phase algorithm for finding the optimal allocation of day-off patterns, while Vohra (1987) provides results on the minimum workforce size required. The three papers consider a somewhat simpler problem than the GCRPWP in that they are only concerned with determining an optimal allocation of days off. In the GCRPWP we must also include the subsequent step of shift allocation if an employee is assigned a particular day on.

One major difficulty with the GCRPWP is the length of the rostering horizon. Six months is far too large to solve in one model. We develop a decomposition approach that breaks the rostering horizon into blocks of manageable size and then solve a sequence of integrated optimization models to construct rosters that span the six-month period. This approach can be seen as a form of the iterative sweeping method described in Eveborn and Rönnqvist (2004). Since excessive solution times are undesirable, to accelerate the solution time for each of the optimization models we implement a heuristic variable fixing routine when forcing integrality. This is a well-known approach (see eg Desaulniers *et al*, 2002; Danna and Pape, 2005). Wäscher and Gau (1996) evaluate several integer fixing heuristics for the cutting stock problem.

When solving practical optimization problems, it is also essential that one includes some degree of robustness in the solution. That is, one should incorporate flexibility in the solution to guard against unexpected uncertainties in the input data. This is becoming an increasingly popular field of research as companies realize the potential savings by not having to re-optimize their schedule when something unforeseen occurs. The majority of recent contributions have appeared in the airline industry (see eg Burke *et al*, 2010; Clausen *et al*, 2010; Weide *et al*, 2010). Here, robustness is incorporated by specifying a certain contingency of over coverage. This is to account for any unexpected increases and/or delays in the forecast workload, and can also compensate for absent employees.

The proposed solution approach has similarities with the set partitioning formulations usually found in the literature on scheduling. Possible solution methods include Mehrotra *et al* (2000), where shift scheduling problems are solved by a branch-and-cut approach, and Eveborn and Rönnqvist (2004), where a branch-and-price approach forms the basis of a practical scheduling system. Branch-and-price has also been applied with success in many cabin crew rostering problems and nurse rostering; see eg Day and Ryan (1997) and Dohn *et al* (2010). However, for these problems, the time horizon is short enough for it to be considered in one model. This is not the case here. Hence, our research contributes to the literature on crew rostering by providing a powerful decomposition method to the GCRPWP, which is also based on branch-and-price and which is capable of finding robust solutions that are proven to be within a few percent of optimality.

3. The GCRPWP

In this section, we consider the GCRPWP in more detail. In particular, we provide a formal definition of the problem and present a cutting stock-based integer programming formulation. To aid in the discussion, we begin by introducing the required terminology and notation.

The GCRPWP entails assigning a set of employees ε (where $|\varepsilon| = n$) to a set of *shifts* $\mathcal{S}$ (indexed from 1, ..., S). The required employee demand on each shift $s \in \mathcal{S}$ must be satisfied as closely as possible. Having too few employees on any given shift is termed *under coverage*, which is undesirable.

In this context, a shift refers to a block of work that has a given start time e_s, a given end time l_s, and a day $d \in \mathcal{D}$ on which the shift starts. Within a shift there are breaks, where no work can be carried out. It is important to account for breaks when calculating the amount of work that can be done within a shift. The set $\mathcal{D}$ (indexed from 1, ..., D) denotes the set of all days in the time horizon, while we denote the set of all shifts on day $d \in \mathcal{D}$ as $\mathcal{S}_d \subseteq \mathcal{S}$. The number of employees required for shift $s \in \mathcal{S}$. is given as q_s. It is assumed that each staff member can perform at most one shift on any given day. In constructing a sequence of shifts for any employee, one must respect several practical constraints. Typically, one must ensure that each employee has a certain minimum rest time between any pair of consecutive shifts and that no employee is assigned more than a certain number of consecutive night shifts. The classification of a shift as being a night shift depends on the shift start time. As stated earlier, it is also required that all employees work the same work pattern. This specifies the number of days an employee will work consecutively, (*on-stretch*), and the number of days of consecutive break, (*off-stretch*). During an off-stretch an employee cannot be assigned any shifts. Here, we consider a 6&3 work pattern. That is, each employee must be assigned six consecutive days (ie six shifts) of work before being assigned a three-day consecutive break. Hence, this is a work pattern of

length nine days. The fixed nature of the work pattern ensures that all employees work, on average, the same number of days per week. A legal sequence of on and off stretches spanning the time horizon gives a *roster-line* for a particular employee or group of employees. The set of roster-lines for all employees together constitutes the *roster*.

The use of a work pattern limits the number of feasible roster-lines; however, the on-stretch is staggered across the employees to ensure an even distribution of off days. One can hence identify a set of *pattern groups* $\mathcal{G}$ based on a given work pattern. Associated with each pattern group is a unique day-off pattern. All day-off patterns conform to the work pattern and state which days of the time horizon an employee will work. For example, an employee could start the time horizon on the first day of the 6&3 work pattern and thus work the first six days of the time horizon, or the employee could start on the eighth day of the work pattern and therefore have the first day of the time horizon off. Naturally, all combinations in between are also possible and this yields nine different pattern groups. Associated with each pattern group $g \in \mathcal{G}$ is an upper bound m_g on the number of employees that can work the corresponding day-off pattern. This can be used, for instance, to force consistency with a previously generated set of roster-lines.

In addition to the hard constraints, which must be respected, it is often desirable to satisfy several soft constraints when constructing rosters. A soft constraint is a constraint that one tries to satisfy if possible; however, it can be violated if necessary. If it is violated, the violation is minimized. An important soft constraint in the GCRPWP is that employees should receive the maximum number of hours off during their three-day break. That is, the first shift of an on-stretch should be a shift starting late in the day, while the last shift should be one finishing early in the day. Inclusion of this soft constraint is described in Section 4.2.

As we stated in Section 2, a key difference between the GCRPWP and many other rostering problems is that the aim is not to find individual roster-lines directly, but rather roster-lines that several employees may work. All staff are assumed to be able to work any shift and due to the 6&3 work pattern, it is impossible to take into consideration such individual preferences as particular weekends on or off. One can hence formally define the GCRPWP as follows: Given a set of employees, a set of shifts (each demanding a certain number of employees), and a work pattern, find an allocation of employees to legal roster-lines such that the total cost of the roster is minimized.

The GCRPWP can be formulated as the following MIP. In what follows, we denote the set of all legal roster-lines as $\mathcal{R}$. We introduce a general integer decision variable x_r for each roster-line $r \in \mathcal{R}$ that counts the number of times roster-line r is used in the solution. We also introduce the binary indicator variables a_{sr} and a_{gr}. The former indicates whether or not shift $s \in \mathcal{S}$ is contained in roster-line $r \in \mathcal{R}$, while the latter indicates whether or not roster-line $r \in \mathcal{R}$ belongs to pattern group $g \in \mathcal{G}$. Additionally, we define a decision variable u_s for each shift $s \in \mathcal{S}$, which indicates the level of under coverage on the shift. A unit of under cover on shift $s \in \mathcal{S}$ is assumed to cost $\check{c}_s$. Finally, we denote the cost of any legal roster-line $r \in \mathcal{R}$ as c_r. This cost reflects the penalties incurred in not satisfying soft constraints.

$$\min \sum_{r \in \mathcal{R}} c_r x_r + \sum_{s \in \mathcal{S}} \check{c}_s u_s, \tag{1}$$

s.t.

$$\sum_{r \in \mathcal{R}} a_{sr} x_r + u_s \geqslant q_s, \qquad \forall s \in \mathcal{S}, \tag{2}$$

$$\sum_{r \in \mathcal{R}} a_{gr} x_r \leqslant m_g, \qquad \forall g \in \mathcal{G}, \tag{3}$$

$$\sum_{r \in \mathcal{R}} x_r \leqslant n, \tag{4}$$

$$u_s \geqslant 0, \qquad \forall s \in \mathcal{S}, \tag{5}$$

$$x_r \in Z_+, \qquad \forall r \in \mathcal{R} \tag{6}$$

The objective function (1) minimizes the total cost of the roster-lines as well as the sum of the penalties incurred in not satisfying the demand of each shift. The first set of constraints (2) ensures that the total demand for any shift is satisfied, possibly through the use of the relevant under cover variable. Constraints (3) restrict the number of roster-lines of a particular pattern group to be no more than the maximum number permitted, while (4) is a constraint on the total number of staff. It prevents one from assigning more roster-lines than there are employees. Constraints (5) and (6) ensure that all decision variables are non-negative. In addition, all x_r variables are also required to be integer. One can observe that the above formulation possesses strong similarities to the well-known cutting stock formulation (see Amor and de Carvalho, 2005). While there are relatively few constraints ($|\mathcal{G}| + |\mathcal{S}| + 1$), there can potentially be an exponential number of variables. In the next section, we briefly introduce the column generation approach for solving problems of this nature, before describing, in detail, how it can be applied to the GCRPWP.

4. Column generation

Column generation is a well-known technique for solving large-scale linear programming problems in which it is impossible to explicitly consider all of the variables in the problem. The approach requires one to decompose the original problem into two optimization problems, which are termed the *master* and the *pricing problem*, respectively.

4.1. The master problem

The master problem is a restricted version of the original problem, containing only a subset of the variables, while the pricing problem is an optimization problem that attempts to identify potential entering variables (columns) for the master problem. The fundamental idea of column generation is that since the majority of the variables in the original problem will be non-basic at an optimal solution, one need only consider, and generate, those variables that have the potential to improve the objective function. The objective function of the pricing problem is hence the reduced cost calculation for the non-basic variables given the dual variable values for the optimal master solution.

Column generation is an iterative procedure between the master and pricing problem. The master problem solves to optimality a restricted version of the original problem and the pricing problem, using the dual variables of the optimal master solution, implicitly prices all non-basic variables and finds the one with the most negative reduced cost. This variable is then added to the master problem. This procedure continues until the pricing problem cannot identify a master variable with negative reduced cost. In which case, optimality of the original problem has been obtained. If one is solving an MIP, integrality can be achieved by embedding the LP column generation methodology in a branch-and-bound framework, termed branch-and-price. For an introduction to column generation, we refer to Desrosiers and Lübbecke (2005).

To apply column generation to the GCRPWP, we first relax the integrality requirements on the x_r variables. That is, constraints (6) are replaced with

$$x_r \in R_+ \qquad \forall r \in \mathcal{R}. \tag{7}$$

The *relaxed master problem* can then be identified as model (1)–(5) and (7). The relaxed master problem with only a subset of the possible roster-lines from $\mathcal{R}$ is termed the *restricted master problem*. Using the dual vector of the optimal solution of the restricted master problem, the role of the pricing problem is to identify the non-basic roster-line with the most negative reduced cost. That is, one must find:

$$\min_{r\in\mathcal{R}} \left(c_r - \sum_{s\in\mathcal{S}} a_{sr}\pi_s - \sum_{g\in\mathcal{G}} a_{gr}\mu g - \gamma \right), \tag{8}$$

where π_s is the dual variable value on the constraint of type (2) associated with shift $s\in\mathcal{S}$, μ_g, is the dual variable value on the constraint of type (3) associated with pattern group $g\in\mathcal{G}$, and γ is the dual variable on Constraint (4). All constraints which define the feasibility of a roster-line must be included in the pricing problem. These include the minimum time that must elapse between successive shifts, the maximum number of consecutive night shifts, and the relevant day-off pattern. How these are enforced is described in Section 4.2. For the GCRPWP, identifying the most negative reduced cost roster in the pricing problem entails solving a resource constrained shortest path problem.

To solve the GCRPWP one must construct a roster that spans a six-month period. Since it is computationally intractable to consider such a long time horizon in one MIP, we present a decomposition approach that divides the six-month time horizon into several shorter blocks, each of which has an associated master problem and a pricing problem. The blocks are solved in sequence, where the solution to any given block is used as input to the subsequent block. In order to be able to easily concatenate the solutions to successive blocks, an overlapping time period, with a duration equal to the number of days in the on-stretch is defined between successive blocks. The duration of the overlap is the shortest that ensures each pattern group has at least one day off in the overlap, or on the day immediately following the overlap. Days off are important since they provide a starting point for which all shift transitions are feasible. For instance, one does not need to remember the information on accumulated night shifts, which would be the case if one was not starting from a day off. Furthermore, proceeding from a day off allows one to more easily enforce the continuation of a pattern group in the pricing problem. At a day off all roster-lines for the same pattern group are essentially in the same state, that is not working. This would not be the case if proceeding from a day on since all roster-lines for the pattern group could potentially be doing different shifts.

Table 1 further illustrates this blocking concept in more detail. Let us assume we are rostering a 6&3 work pattern, that the first block runs from day 1 to day 18 and that the second block begins on day 13. The overlap spans days 13 to 18. The days on for each pattern group are marked with a one, while a zero indicates a day off. One can see that all but pattern group 7 have a day off during the overlap. Day 19 for this group, however, is a day off.

For each pattern group, one would like the pricing problem of the second block to start from the day after its last day off in the overlap. For example, the pricing problem for pattern group 4 should be defined from day 16, while that of pattern group 6 should be defined from day 14. Since the solution to the first block produces a roster for days 1 to 18, to achieve this one can keep the solution as it is for all days up until the last day off in the overlap for each pattern group and resolve all other days. Enforcing part of the solution from the first block entails fixing all shifts up until the last day off in the overlap for each pattern group. In Table 1, the (pattern group, day) combinations marked with grey give the shifts that will be fixed in the second block. For example, the shifts in the roster-lines assigned to pattern group 3 in block 1 are fixed on day 13, but resolved for days 17 and 18 in the

Table 1 Block overlap and shift fixing

	Days																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Group 1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1
Group 2	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1
Group 3	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1
Group 4	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1
Group 5	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1
Group 6	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
Group 7	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0
Group 8	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0
Group 9	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0

optimization of the second block. To fix a shift in a particular block, one simply reduces the right-hand side of the relevant constant (2) by the number of roster-lines covering it. This resolving step also allows one to correct for any bad choices made as a result of optimizing over a time horizon that is too short.

In summary, to concatenate blocks efficiently, one can roll back to the last day off in the overlap for each pattern group, fix all shifts assigned up until this point, and then resolve all later days in the optimization of the subsequent block.

4.2. Pricing problem

As described previously, the pricing problem determines whether any columns with negative reduced costs exist. If such columns exist, then these must be generated. In this paper, we will use a pricing problem for each pattern group as it makes the identification of legal roster-lines easier. To identify legal roster-lines for a given pattern group, we construct a directed acyclic graph, where the nodes represent possible activities and the arcs represent transitions between activities. An activity can be to work a shift on a certain day or to have the day off. With some additional constraints described below, the identification of a legal roster-line amounts to solving a resource constrained shortest path problem in an acyclic graph. A feasible roster-line must satisfy the following hard constraints:

1. no staff member can perform more than one shift on any day;
2. an employee must have at least 10 h of rest between consecutive shifts;
3. given pattern group $g \in \mathcal{G}$, an employee has to work the day-off pattern corresponding to that pattern group;
4. no more than three consecutive night shifts are allowed.

Constraints 1–3 can be handled in the construction of the directed acyclic graph, whereas Constraint 4 must be enforced using a resource. In addition to the hard constraints, we also have the following soft constraints

5. the first shift on an on-stretch should be a late shift;
6. the last shift on an on-stretch should be an early shift.

The soft constraints are handled in the objective function of the pricing problem and will be discussed later.

Before describing the pricing algorithm in detail, we first expand the notation. We let $\mathcal{S}' = \mathcal{S}\{S+1, \ldots, S+D\}$ be the index set of activities, where activity $S+d$ corresponds to having day d off. Hence, we let $\mathcal{S}'_d = \mathcal{S}_d\{S+d\}$ be the possible activities on day d. To be able to distinguish between night shifts and the remaining shifts, we define $\mathcal{N} \subseteq \mathcal{S}$ as the index set of night shifts. We let $v = (v_0, \ldots, v_p)$ be a binary vector of length $(p+1)$ corresponding to the number of days in the work pattern. For example, v has length 9 when applying a 6&3 work pattern. Each entry in v indicates whether the day is a day on or a day off. For instance $v = (1,1,1,1,1,1,0,0,0)$ specifies a day-off pattern and states that the six first days is the on-stretch and the last three days is the off-stretch. Since we may need to use the pricing problem in different settings, for example for different day intervals and day-off patterns, we construct a representation which is sufficiently general to accommodate the required settings.

The individual roster-lines are sequences of on-stretches and off-stretches. Each activity has a time interval of execution and this gives a natural ordering of activities, where some activities are successors of others. This ordering gives rise to an acyclic directed graph, which we refer to as the underlying graph. In the following, we denote ω as the origin and δ as the destination. For a given day interval $[d_1, d_2]$ we denote

$$\mathcal{V}'(d_1, d_2) = \bigcup_{d=d_1}^{d_2} \mathcal{S}'_d$$

as the index set of shift nodes joined with the index set of the day-off nodes for all days between d_1 and d_2. The set of

vertices of the graph will then be $\mathcal{V}(d_1, d_2) = \{\omega, \delta\} \bigcup \mathcal{V}'(d_1, d_2)$.

We will refer to $\mathcal{A}(d_1, d_2)$ as the set of arcs in the graph. On each day, exactly one activity has to be carried out. This constitutes layers in the graph, that is one layer for each day d in the interval $[d_1, d_2]$. In the set of nodes $\mathcal{V}'(d_1, d_2)$ it is only possible to progress from one day to the next day, thus only arcs between nodes $i \in \mathcal{S}'_d$ and $j \in \mathcal{S}'_{d+1}$ for $d = d_1, \ldots, d_2 - 1$ are included. The origin ω only has arcs leaving it. For shift fixing, we need to be able to fix activities for days d_1 to day h where $h \leqslant d_2$. To make this possible, we include an arc from the origin to all nodes $i \in \mathcal{V}'(d_1, d_2)$ Later, we will eliminate the arcs which are not allowed from the origin. The destination node δ only has entering arcs. As we have to have exactly one activity each day, it is only possible to enter the destination from the last day in the interval. Hence, the only arcs (i, δ) that are allowed to enter the destination are those with $i \in \mathcal{S}\prime_{d_2}$. Note that we also eliminate any transition between two shifts that does not satisfy the 10-h rule.

In Figure 1 we give an example of the underlying graph for the day interval [20, 28]. Each of the days has a column of nodes corresponding to the possible activities on that day. The black nodes are day-off nodes, the grey nodes are the night-shift nodes and the white nodes are the remaining shifts. The nodes are therefore partitioned horizontally. Each partition may have multiple nodes, of each kind, in each column, but for simplicity we have only shown one node. The transitions between activities are shown as arcs between the layers. The black arcs are those which are penalized due to soft constraint violations, whereas the grey arcs have a cost of zero. The difference between dashed arcs and filled arcs is explained later.

A path $P = (w_0, \ldots, w_p)$ from the origin $w_0 = \omega$ to the destination $w_p = \delta$ can be translated directly into a roster-line (and vice versa) as the nodes in the path correspond to individual activities. Furthermore, any path will have at most one shift each day and will satisfy the 10-h rule. We still need to satisfy the constraints for each pattern group and the requirement that no employee can have more than three consecutive night shifts.

One can easily apply a day-off pattern to the underlying graph. With any arc $(i, j) \in \mathcal{A}(d_1, d_2)$ we associate a binary value $u(i, j)$, which is equal to 1, if and only if, we allow the arc to be used in the solution. We put the value of $u(i, j) = 1$ for all arcs $(i, j) \in \mathcal{A}(d_1, d_2)$ unless it is stated otherwise. This value allows us to use the same underlying graph for several different setups of the pricing problem.

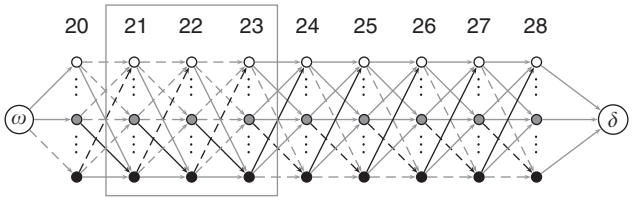

Figure 1 Example of 6&3 pattern.

First, we use the $u(i, j)$ to apply the pattern groups. Suppose that we are given $v = (v_0, \ldots, v_p)$ and wish to apply this as a day-off pattern. We assume that the day-off pattern is applied from the first day index of $\mathcal{D}$, that is v_0 states whether day 0 should be a work day or not. In general, given a day $d \in \mathcal{D}$ we have that $v_{(d \bmod (p+1))}$ states whether or not day d is a day on or a day off. Now, for each day $d \in [d_1, d_2]$ we have the following two cases:

1. if $v_{(d \bmod (p+1))} = 1$ then day d is a day on and it should not be allowed to enter the day-off node $m + d$ for day d. Hence, for all arcs $(i, m + d) \in \mathcal{A}(d_1, d_2)$ we set $u(i, m + d) = 0$;
2. if $v_{(d \bmod (p+1))} = 0$ then day d is a day off and it should not be allowed to enter any shift nodes $j \in \mathcal{S}_d$. Hence, for all arcs $(i, j) \in \mathcal{A}(d_1, d_2)$ with $j \in \mathcal{S}_d$ we set $u(i, j) = 0$.

Hence, we can modify the underlying graph to satisfy any day-off pattern vector. Given that d is the first day which is not fixed, we put $u(\omega, j) = 0$ for all $j \notin \mathcal{S}'_d$. This ensures that we can only visit nodes in $\mathcal{S}'_d$ as the first node after the origin.

In Figure 1, we have applied the day-off pattern $(v_0, v_1, \ldots, v_8) = (1, 1, 1, 0, 0, 0, 1, 1, 1)$. The dashed arcs are those which have to be eliminated to accommodate the day-off pattern. According to this, day 20 will be a day on as $v_{(20 \bmod 9)} = v_2 = 1$ and day 21 will be the first day off in the off stretch. The frame around the nodes for days 21–23 indicates that these days are days off. Note that we have not eliminated all unnecessary arcs, but only a sufficient subset of arcs to enforce the day-off pattern. A more substantial elimination is possible, but in practice it does not have a significant impact on the running time of the algorithm.

The direct cost of a roster is based on the penalties given for violating the soft constraints. We assume that the soft constraints are constraints on individual transitions between shifts. Let c_{ij} be the penalty of using arc (i, j). Let λ_{ij} be the dual price of using arc (i, j). The reduced cost accumulated along path P_r is

$$\bar{c}(P_r) = \sum_{q=0}^{p-1} \left(c_{(w^r_q, w^r_{q+1})} - \lambda_{(w_q, w_{q+1})} \right).$$

The objective of the pricing problem is given in problem (8), in which we have $\lambda_{ij} = \pi_j$ for all $(i, j) \in \mathcal{A}(d_1, d_2)$ with $j \in S_d$ and $\lambda_{ij} = 0$ for all $(i, j) \in \mathcal{A}(d_1, d_2)$ with $j \notin S$. Since all pattern groups $g \in \mathcal{G}$ are independent, for all roster-lines $r \in \mathcal{R}$ we have that $a_{gr} = 1$ for exactly one pattern group $g \in \mathcal{G}$ and $a_{gr} = 0$ for all other pattern groups. Furthermore, as there are only a small number of pattern groups, for example nine for the 6&3 work pattern, we may keep the

pattern group fixed and solve one problem for each. For each pattern group $g \in \mathcal{G}$ we have the following decision problem:

$$\min_{r \in \mathcal{R}: a_{gr}=1} \bar{c}(P_r) - \mu_g - \gamma < 0, \tag{9}$$

which checks whether or not a negative reduced cost roster-line exists for the pattern group.

Once we have the underlying graph, we can try to identify feasible (ω, δ)-paths. Such a path will correspond to a roster satisfying the day-off pattern υ. The GCRPWP has, however, an additional restriction for a roster-line, which cannot be handled directly by modifying the underlying graph, and for which we use the notion of resources and resource extension functions. We refer the reader to Desaulniers *et al* (1998), Irnich and Desaulniers (2005), and Irnich (2008) for an introduction to resource extension functions. The requirement that no more than three consecutive night shifts is modelled by a resource. The intuition is that the resource is initialized at zero and is incremented by one each time a night shift is undertaken. When the accumulated resource is equal to three, then it should not be possible to transition to another night shift. As it is not the total number of night shifts which is bounded but the number of consecutive night shifts, we have to reinitialize the resource at zero whenever a non-night-shift activity is undertaken. The resource extension function has the property that given two paths P and P' both ending in node i with a resource consumption of T_i and T_i', respectively, such that $T_i \leqslant T_i'$, then for any extension of P', we can identify a similar extension of P with at most the same consumption as the extension of P'. Thus, we will consider P to be a better path than P' with respect to the resource.

To identify a cost-minimizing and resource-feasible path in the underlying graph, we turn to dynamic programming. This type of problem is often referred to as a shortest path problem with resource constraints. The fundamental idea is to construct paths by extending partial paths to all possible successor nodes and do this repeatedly until no path can be extended. If this is done carefully, we will end up with at least one cost-minimizing and resource feasible path. We briefly describe the dynamic programming procedure we use, but refer the reader to Irnich and Desaulniers (2005) for a review of dynamic programming algorithms for resource constrained shortest path problems. The structure of the pricing problem will allow us to solve the problem efficiently.

To each path P we associate a state (or a label) $\mathcal{L}(P) = (\bar{c}(P), T(P))$ which is the vector of accumulated reduced cost and the number of current consecutive night shifts. Given two paths P and P', both ending in node i, we would like to determine which one is the most promising. If the two states $\mathcal{L}(P) \neq \mathcal{L}(P')$ are distinct and, in addition, we have $\bar{c}(P) \leqslant \bar{c}(P')$ and $T(P) \leqslant T(P')$, then the path P is the most promising. We say that the path P dominates the path P' and write $\mathcal{L}(P) \prec \mathcal{L}(P')$. In this case, we call P the dominant path and P' the dominated path. When a dominated path is extended, the resulting path will also be dominated by an identical extension of the dominant path. Hence, we can eliminate the dominated path. For any node i, we let F_i be the set of all known states with paths ending in node i. Furthermore, we let $E_i = \{\mathcal{L}(P) \in F_i | \nexists \mathcal{L}(P'): \mathcal{L}(P') \prec \mathcal{L}(P)\}$ be the set of efficient states. It is sufficient to extend the paths having states in E_i as the paths with states in F_i / E_i will be dominated by at least one path in E_i. From the layers of the graph, the nodes have an inherent topological order. Hence, we need only to extend paths from each node once, when using the ordering $(n_1, \ldots, n_{|V|})$. Thus we have a natural iterative approach for the dynamic programming, where we iterate through the nodes given their topological ordering.

The acyclic resource constraint shortest path problem can be solved in pseudo-polynomial time (Desrochers and Soumis, 1988). The algorithm is pseudo-polynomial on the resource width, that is on the number of feasible values of the resources. As our pricing problem has only one resource and as that resource is bounded by the work pattern length, the pricing problem can actually be solved in polynomial time for a given day-off pattern.

5. Enforcing integrality of the solution

As mentioned earlier, column generation is used to solve the relaxed master problem. However, to solve the GCRPWP, we need a solution to the original master problem and hence we reintroduce the integer requirement for variables x_r, where it is violated. The traditional approach to reintroduce integrality is by including the column generation procedure in a branch-and-price framework. In a standard MIP-model, variable branching is usually the branching method of choice. In variable branching the solution space is partitioned into two disjoint subspaces (branches), constructed by splitting the value set for a single variable with a current fractional value $x_b = x_b^*$. In the left branch, the variable is bounded downwards, that is $x_b \leqslant \lfloor x_b^* \rfloor$, and in the right branch it is bounded upwards, ie $x_b \geqslant \lceil x_b^* \rceil$. Unfortunately, it is hard to transfer this approach directly to a column generation context. In column generation, the property that keeps us from regenerating existing columns is the fact that any variable in an optimal basic solution has a reduced cost of zero and all existing non-basic variables have non-negative reduced costs. This is not true for variables with an upper bound. These may have negative reduced cost, even if they are in the basis. The variable bound may instead be enforced by an additional constraint in the master problem. The dual of the new constraint would be reflected in the

reduced cost of the variable, and the variable would therefore be non-negative in an optimal solution. However, in the pricing problem, it is not trivial and usually highly inefficient to deal with dual costs for specific variables.

As variable branching is not well suited for column generation, another approach is usually taken, namely the use of constraint branching. For set partitioning, set packing, set covering, and bin packing problems, the approach introduced by Ryan and Foster (1981) is widely used. This approach requires the two constraints that define the constraint branch to have unit right-hand sides. Unfortunately, it does not carry over to the generalized set covering problem, which we are considering here. Desaulniers (2010) proposes a four-layered branching strategy to the split delivery vehicle routing problem, where the master problem is similar to that of the GCRPWP. An assumption for the completeness of the branching strategy is that there is only one, so called, split customer. In the split delivering vehicle routing problem this assumption always holds, but we do not have the corresponding property in the GCRPWP. The branching strategy may be applied, but as it is not complete, there may be fractional solutions where no branching candidate exists. For a complete branching strategy, we consider another related problem, namely the cutting stock problem. Amor and de Carvalho (2005) describe a branching strategy for a model similar to the master problem presented here. Branching is applied to aggregated arc flows. To get a complete branching scheme, nodes of the pricing problem may need to be split into several nodes.

Rostering problems contain a high level of degeneracy and as a consequence, it is often possible to find many different optimal solutions. Indeed, initial tests on the GCRPWP showed that this was the case, and the fractionality of solutions did usually only decrease after introducing a very large number of branching constraints. As described above, it is theoretically possible to find the optimal solution for any instance of the GCRPWP by exploring the full branch-and-bound tree. However, as small run times are desired here, instead we introduce a greedy approach to achieve integrality.

From a fractional solution, the idea is to iteratively apply variable fixing to the variables in the optimal LP-solution. As described above, in column generation, it is computationally hard to bound variables with an upper bound. Therefore, we introduce solely lower bounds on the variables. The bounding will in most cases have an identical effect to that of true variable fixing.

In the following, we describe the variable bounding scheme. Consider an optimal solution to the relaxed master problem, x^*. If $x_r^* \in \mathbb{Z}_+, \forall r \in \mathcal{R}$ then x^* is also a solution to the original formulation and the algorithm terminates. Otherwise, we look at the fractional part of the variable values, $f_r^* = x_r^* - \lfloor x_r^* \rfloor$. Given a pre-specified threshold, τ (with $0 < \tau \leqslant 1$), we impose the following bounds:

$$x_r \lceil x_r^* \rceil, \qquad \forall r \in \mathcal{R} : f_r^* \tau. \tag{10}$$

If $f_r^* < \tau$ for all $r \in \mathcal{R}$, we instead impose the bound on the variable with the largest fractional part:

$$\{r \in \mathcal{R} : f_r^* \tau\} = \emptyset \Rightarrow x_r \lceil x_r^* \rceil,\ r = \arg\max_r f_r^*. \tag{11}$$

As we can introduce an additional bound for any fractional solution and as the value of any variable in an optimal solution of the GCRPWP is finite, this approach eventually gives a feasible integer solution, assuming that m_g is integer. The approach may be seen as a special case of variable branching, where the left branch is never explored.

6. Rostering directly on the forecast workload

In Section 3, the mathematical model of the GCRPWP was introduced. The model assumes that the employee demand has been defined for each shift, that is q_s is defined for any shift, $s \in \mathcal{S}_d$. Determining the value of q_s is an optimization problem in itself, but we have so far assumed that it is predetermined, usually by an experienced manual planner.

Figure 2 shows a workload graph over one day. The light grey area is the forecast workload discretized in 5-min intervals. The dashed bold line shows the suggested shift cover (denoted as Cover-B in Figure 2). Since shifts contain breaks, the actual cover is sometimes lower than the shift cover. The actual cover is depicted with the full bold line, or the dark grey area in the figure. For each time interval, the cover has been found by summing the demands of all shifts that overlap with that time interval.

In the following, we introduce a model, where the roster is constructed to directly cover the forecast workload. As a consequence, we are going to disregard the values of q_s and the process of defining shift demands becomes superfluous. Figure 3 shows how one step of the usual rostering workflow is circumvented with this approach.

We introduce a discretization of time and refer to an individual time interval in the set of intervals as $t \in \mathcal{T}$. The set $\mathcal{S}_t \subseteq \mathcal{S}$ contains all shifts that overlap with time interval t. The forecast workload of period t is denoted wt. $\check{c}_t$ refers to the cost of under coverage in interval t. The mathematical model becomes:

$$\min \sum_{r \in \mathcal{R}} c_r x_r + \sum_{t \in \mathcal{T}} \check{c}_t u_t, \tag{12}$$

s.t.

$$\sum_{s \in \mathcal{S}} \sum_{r \in \mathcal{R}} a_{sr} x_r + u_t \geqslant w_t, \qquad \forall t \in \mathcal{T}, \tag{13}$$

$$\sum_{r \in \mathcal{R}} a_{gr} x_r \leqslant m_g, \qquad \forall g \in \mathcal{G}, \tag{14}$$

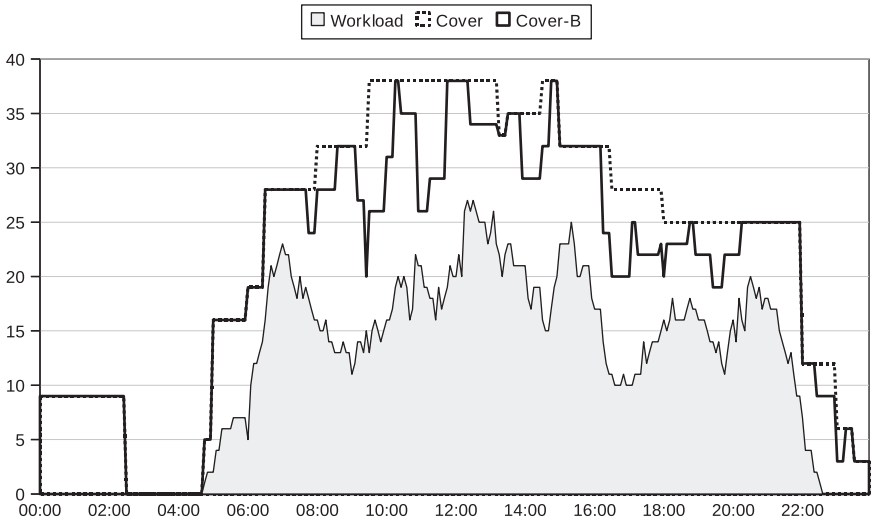

Figure 2 A workload graph with a suggested shift cover.

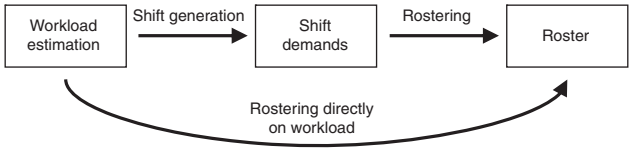

Figure 3 The workflow in rostering, where the intermediate step of creating shift demands is made superfluous by rostering directly on the workload estimation.

$$\sum_{r\in\mathcal{R}} x_r \leqslant n, \tag{15}$$

$$u_s \geqslant 0, \qquad \forall t \in \mathcal{T}, \tag{16}$$

$$x_r \in Z_+, \qquad \forall r \in \mathcal{R} \tag{17}$$

The model is similar to that of Section 3. The objective (12) sums the cost of under coverage for all intervals. For each interval, the workload is either fully covered or the corresponding under coverage is registered in u_t (13). Constraints (14)–(17) correspond directly to Constraints (3)–(6).

An issue with this model is the number of constraints of the form (13), as the set $\mathcal{T}$ may be very large. To alleviate this problem, we aggregate the time intervals. In the following, we describe how to do this without losing any information in the model.

The idea is to aggregate all time intervals for which the cover is always equal. The covers of two separate time intervals are equal if they overlap with an identical set of shifts. Let $\mathcal{T}_1, \ldots, \mathcal{T}_q$ refer to a partitioning of time intervals in q partitions, where all time intervals in a set $\mathcal{T}_v$ have the same shift cover. The partitioning is made so that $\mathcal{T}_1 \bigcup, \ldots, \bigcup\ \mathcal{T}_q = \mathcal{T}$ and for any two sets $\mathcal{T}_i$ and $\mathcal{T}_j: i \neq j \Rightarrow \mathcal{T}_i \bigcap \mathcal{T}_j = \emptyset$. Any set can hold only consecutive elements. As explained in Section 3, each shift contains a break and the cover of a shift does therefore not contain a set of consecutive time intervals. In this work, we assume that each shift contains only one break, starting at time e_s^b ending at l_s^b. The model is easily extended to consider multiple breaks.

The partitioning into these sets is straight forward. Let the set $\mathcal{T}^p = \{t_1^p, t_2^p, \ldots, t_q^p, t_{q+1}^p\} = \bigcup_{s\in\mathcal{S}} \{e_s, l_s, e_s^b, l_s^b\}$ contain all split times in sorted order. We define $\mathcal{T}_i = t_1^p, \ldots, t_{q+1}^p - 1, i = 1, \ldots, q$. From the partitioning of $\mathcal{T}$ define a new set of time intervals $\Theta = \theta_1, \ldots, \theta q$, where the start times and the end times of the new time intervals are the split times of $\mathcal{T}^p$. The definition of $\mathcal{S}_t$ easily transfers to $\mathcal{S}_\theta$ for $\theta \in \Theta$. However, the workload in a time intervals θ_i is not necessarily constant and hence the under cover is not as easily defined as in the previous model.

Let $w_{\theta_i} = \max_{t\in\mathcal{T}_i} w_t$ and introduce the decision variables $u_{\theta_{ij}}, i = 1, \ldots, q, j = 1, \ldots, w_{\theta_{ij}}$ with $0 \leqslant u_{\theta_{ij}} \leqslant 1$. $\Sigma_{j=1}^{w_{\theta_i}} u_{\theta_{ij}}$ denotes the under coverage for time interval θ_i. By defining the under coverage as a sum of variables, we are able to introduce a piecewise linear cost function for under coverage. The cost of each piece of the function is defined by the number of original time intervals, for which the cover is insufficient. The cost of $u_{\theta_{ij}}$ is $\check{c}_{\theta_{ij}}$ and is calculated as:

$$\check{c}_{\theta ij} = \sum_{t\in\mathcal{T}_i : w_t > w_{\theta i} - j} \check{c}_t. \tag{18}$$

$\check{c}_{\theta_{ij}}$ sums the cost of all original time periods, which will not be fully covered if the interval θ_i has under coverage of j or more.

Figure 4 shows a workload graph zoomed in on one interval, θ', covering the time from 15:00 to 16:00. The grey

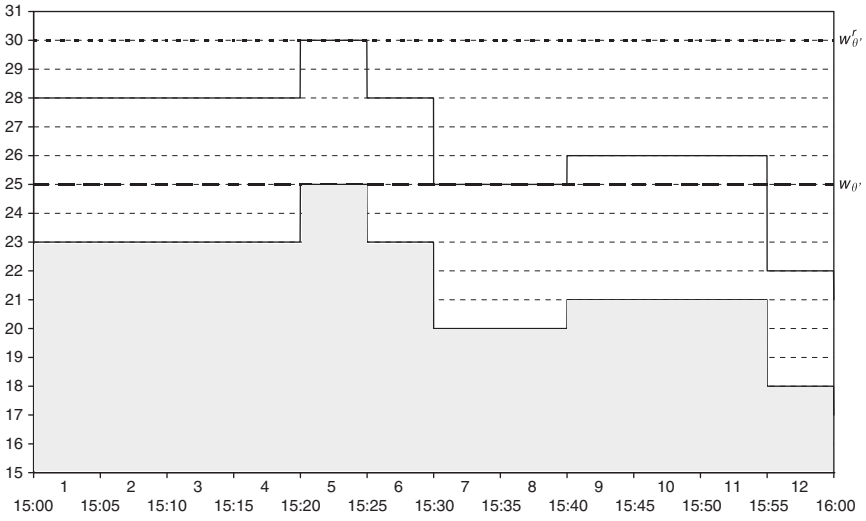

Figure 4 Example of cost calculation for aggregated time intervals.

area is the workload. The curve above the workload area with a shape similar to the workload represents robustness, which is described in Section 7. We disregard this right now. In the example, $w_{\theta'} = 25$. As an under coverage of 1 (relative to $w_{\theta'}$) only introduces under coverage in interval 5: $\check{c}_{\theta' 1} = \check{c}_5$. Following the same argument, $\check{c}_{\theta' 2} = \check{c}_5$. Under coverage of 3 or 4 introduces under coverage in more intervals and therefore: $\check{c}_{\theta' 3} = \check{c}_{\theta' 4} = \check{c}_1 + \check{c}_2 + \check{c}_3 + \check{c}_4 + \check{c}_5 + \check{c}_6$. The cost coefficients for further under coverage are calculated similarly. Hence, the aggregated model becomes:

$$\min \sum_{r \in \mathcal{R}} c_r x_r + \sum_{\theta \in \Theta} \sum_{j=1}^{w_\theta} \check{c}_{\theta j} u_{\theta j}, \tag{19}$$

s.t.

$$\sum_{s \in \mathcal{S}_\theta} \sum_{r \in \mathcal{R}} a_{sr} x_r + \sum_{j=1}^{w_\theta} u_{\theta j} \geqslant w_\theta, \qquad \forall \theta \in \Theta, \tag{20}$$

$$\sum_{r \in \mathcal{R}} a_{gr} x_r \leqslant m_g, \qquad \forall g \in \mathcal{G}, \tag{21}$$

$$\sum_{r \in \mathcal{R}} x_r \leqslant n, \tag{22}$$

$$0 \leqslant u_{\theta j} \leqslant 1, \qquad \forall \theta \in \Theta,\ \forall j = 1, \ldots, w_\theta, \tag{23}$$

$$x_r \in Z_+, \qquad \forall r \in \mathcal{R} \tag{24}$$

The constraints of the model correspond directly to Constraints (12)–(17) with the described aggregation, $\check{c}_{\theta_{ij}}$ is increasing over j and therefore, in an optimal solution, we have that $u_{\theta 1} \geqslant u_{\theta 2} \geqslant \cdots \geqslant u_{\theta w_\theta}\ u_{\theta w \theta}$, as intended. At most one u_{θ_j} is fractional. If w_θ is integer, all u_{θ_j} are binary.

In practice, we want to limit the number of variables as much as possible, $\check{c}_{\theta_{ij}}$ is often unchanged for a sequence of values of j. In this case, we may remove all but one of the associated variables by increasing the upper bound of the remaining variable, accordingly.

7. Robustness

In the previous section, we assumed that there was no preference between rosters that cover the forecast workload equally well. In practice, the actual workload on a given day is not going to match the forecast workload exactly, and we therefore introduce robustness measures to create a roster, which deals well with small changes in workload.

To be able to handle a larger workload than anticipated, we add a certain percentage to the original forecast workload. We refer to this contingency as r^c, eg $r^c = 0.2$ for a 20% contingency. Furthermore, we expect some tasks to be delayed. This will result in a delayed workload. We define r^d as the number of minutes of slippage that need to be accounted for, for example $r^d = 15$. We want to cover both cases as well as possible.

Let $\check{c}^r_t$ be the cost per unit of not covering the workload with the added contingency or of not covering the slipped workload, whatever is more demanding. The objective of the disaggregated model (12) is changed and two additional constraints are added:

$$\min \sum_{r \in \mathcal{R}} c_r x_r + \sum_{t \in \mathcal{T}} (\check{c}_t u_t + \check{c}^r_t u^r_t), \tag{25}$$

$$\sum_{s \in \mathcal{S}_t} \sum_{r \in \mathcal{R}} a_{sr} x_r + u_t + u^r_t \geqslant \lceil (1 + r^c) w_t \rceil, \qquad \forall t \in \mathcal{T}, \tag{26}$$

$$\sum_{s \in \mathcal{S}_t} \sum_{r \in \mathcal{R}} a_{sr} x_r + u_t + u^r_t \geqslant w_{t - r^d}, \qquad \forall t \in \mathcal{T}. \tag{27}$$

A new set of variables, u^r_t, has been introduced to correctly penalize inadequate robustness. The right-hand

side of Constraints (26) is converted to an integer for simplicity. This means that under coverage is always integer if the workload estimations are.

The changes to the model carry over to the aggregated model easily. Each time interval may now contribute to the cost of the aggregation with either $\check{c}_t$ or $\check{c}_t^r$. The right-hand sides of Constraints (20) hold the necessary coverage for no penalty to apply, including penalties from robustness. Hence, we define: $w_{\theta_i}^r = \max_{t\in\mathcal{T}_i}(\max\{(1+r^c)w_t,\ w_{t-r^d}\})$. We also introduce the subset $\mathcal{T}_{ij}^r = \{t \in \mathcal{T}_i : \lceil(1+r^c)w_t\rceil > w_{\theta_i}^r - j \vee w_{t-r^d} > w_{\theta_i}^r - j\}$. The right-hand side of (20) is replaced by $w_{\theta i}^r$ and the coefficients of the objective function become ($i=1,\ \ldots,q, j=1,\ \ldots,\ w_{\theta_i}^r$):

$$\check{c}_{\theta ij}^r = \sum_{t\in\mathcal{T}_i:w_t>w_{\theta_i}^r-j} \check{c}_t + \sum_{t\in\mathcal{T}_{ij}^r} \check{c}_t^r. \tag{28}$$

As a consequence, the right-hand sides of (20) are increased, but the costs are decreased correspondingly. For $\check{c}_t^{\prime r}=0$ the total cost is equal to the cost of the model without robustness. Assuming that $\check{c}_t^{\prime r} \leqslant \check{c}_t$, in an optimal solution we still have $u_{\theta 1} \geqslant u_{\theta 2} \geqslant \cdots \geqslant,\ u_{\theta w_\theta^r}$ with no fractional u_{θ_j} for integer w_θ^r.

As an example, refer again to Figure 4. Under coverage is now relative to $w_{\theta'}^r$. Again, an under coverage of 1 or 2 (relative to $w_{\theta'}^r$) only introduces under coverage in interval 5 and therefore: $\check{c}_{\theta' 1} = \check{c}_{\theta' 2} = \check{c}_5^r$. The subsequent coefficients are calculated as: $\check{c}_{\theta' 3} = \check{c}_{\theta' 4} = \check{c}_1^r + \check{c}_2^r + \check{c}_3^r + \check{c}_4^r + \check{c}_5^r + \check{c}_6^r$, $\check{c}_{\theta' 5} = \check{c}_1^r + \check{c}_2^r + \check{c}_3^r + \check{c}_4^r + \check{c}_5^r + \check{c}_6^r + \check{c}_9^r + \check{c}_{10}^r + \check{c}_{11}^r$. The coefficient of an under coverage of six includes the original costs as well: $\check{c}_{\theta' 6} = \check{c}_1^r + \cdots + \check{c}_{11}^r + \check{c}_5$. The calculations for the remaining coefficients are similar. The final model with time interval aggregation and robustness becomes:

$$\min \sum_{r\in\mathcal{R}} c_r x_r + \sum_{\theta\in\Theta}\sum_{j=1}^{w_\theta^r} \check{c}_{\theta j}^r u_{\theta j}, \tag{29}$$

s.t.

$$\sum_{s\in\mathcal{S}_\theta}\sum_{r\in\mathcal{R}} a_{sr}x_r + \sum_{j=1}^{w_\theta^r} u_{\theta j} \geqslant w_\theta^r, \qquad \forall\theta\in\Theta, \tag{30}$$

$$\sum_{r\in\mathcal{R}} a_{gr}x_r \leqslant m_g, \qquad \forall g\in\mathcal{G}, \tag{31}$$

$$\sum_{r\in\mathcal{R}} x_r \leqslant n, \tag{32}$$

$$0 \leqslant u_{\theta j} \leqslant 1, \qquad \forall\theta\in\Theta, \forall j=1,\ \ldots,\ w_\theta^r, \tag{33}$$

$$x_r \in Z_+, \qquad \forall r\in\mathcal{R} \tag{34}$$

8. Experimental results

In this section, we present the results obtained for the proposed methodology on three real-life instances supplied by the airline. Owing to its superiority from a robustness modelling perspective, we only test Model (29)–(34). The instances, denoted W07, S08, and S10 below, each have a rostering horizon of 189 days and contain 65, 95, and 139 staff members, respectively. All instances have 11 different shifts, three of which are night shifts. To incorporate flexibility with respect to breaks, three copies of each shift are created and differ only in the break time of the shift. The workload demand is cyclic with a period of one week and all staff are assumed to be working a 6&3 work pattern.

Since this is a somewhat small test set, an additional 10 artificial instances have been constructed and used in the analysis of the algorithm's performance. All artificial instances are based on the real-life instances and are, in particular, an attempt to stress test the approach given more dramatic workload demands. That is, the artificial instances are identical in structure to the real-life instances in terms of the number of shifts and rostering horizon; however, each has a different workload demand and number of staff. These instances are referred to as A01–A10 below. The entire algorithm has been written in the C++ programming language and utilizes the commercial solver Cplex 10.0 with default parameters to solve the master problem. All computational experiments have been performed on a 64-bit Linux operating system equipped with a dual core AMD 2.2 GHz processor and 2GB of RAM.

We begin with an analysis of how the model and solution approach perform over a 63-day time horizon. Before considering a longer time horizon, we want to ascertain the effect on solution quality of decomposing the rostering horizon into shorter, more tractable, overlapping blocks. Also the heuristic shift fixing and branching routines described in Sections 4 and 5 will compromise solution quality. The question is, to what degree. In the 63-day time horizon, we are able to calculate the optimal solution to the LP relaxation of the non-decomposed problem and thus provide a lower bound on the value of the decomposed integer solution obtained. The non-decomposed model is also of the form (29)–(34) and solved using the same methodology; however, it is solved as one block as opposed to multiple blocks in the decomposed case. Furthermore, as 63 is the lowest common multiple of the work pattern length and the period of the workload demand, if the obtained rosters are *wrappable*, then one can simply copy the rosters to any rostering horizon that is a multiple of 63 days. A wrappable roster is one in which it is possible for all staff on a particular pattern group to transition from their last shift (on day 63) to one of the shifts worked by the pattern group on the first day. One cannot guarantee this to always be the case, since such shift transitions are not taken into consideration in our approach.

To test our decomposition approach the 63-day horizon is divided into an initial block of 23 days and five additional blocks with a length of 14 days. Each additional

Table 2 Results for a 63-day rostering horizon

Instance	LP^{I}	IP^{I}	LP^{D}	IP^{D}	LP^{N}	G^{D} (%)	G^{N} (%)	$t^{D}_{IP}(s)$	$t^{N}_{LP}(s)$
A01	59.18	59.18	39.90	39.90	39.90	0.00	0.00	110.67	614.94
A02	375.18	377.89	251.43	254.14	250.36	1.08	1.51	805.84	2516.20
A03	365.20	366.96	241.55	243.31	242.17	0.73	0.47	692.48	4783.25
A04	740.31	740.51	495.93	496.12	495.97	0.04	0.03	236.87	1316.60
A05	327.78	328.17	218.43	218.82	218.57	0.18	0.11	1122.08	3041.76
A06	7182.80	7183.84	4799.55	4800.59	4799.94	0.02	0.01	711.48	2933.60
A07	0.55	0.55	0.35	0.35	0.35	0.00	0.00	27.29	204.02
A08	1630.88	1631.00	1099.57	1099.69	1099.60	0.01	0.01	130.69	546.72
A09	232.15	237.60	150.41	155.86	152.07	3.62	2.49	2079.08	6378.47
A10	4595.07	4597.57	3171.40	3173.90	3172.17	0.08	0.05	713.56	2211.54
S08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	19.05	45.48
S10	306.68	306.83	217.46	217.61	217.48	0.07	0.06	147.92	965.84
W07	0.85	0.85	0.57	0.57	0.57	0.00	0.00	49.24	305.71

block contains eight new days and six days of the preceding block (that will be resolved). For this discretization one has many possibilities; we tried various values, and the above parameters appeared to work well. In all experiments the cost of 1 h of uncovered work is assumed to be 10 units, while an hour of uncovered robustness costs 1 unit. Given that we are dealing with a cyclic, weekly workload demand and that staff are working a fixed 6&3 work pattern, a pre-allocation step that equalizes the number of staff working each pattern group is performed. That is, the algorithm does not determine how many staff members will be in each pattern group. In all our experiments τ is set to 1.0, thus ensuring we fix only one variable at each branching step. Namely, the variable with the largest fractional component.

Table 2 provides the results of the initial experiments. For each instance, the table gives the *inflated* LP and IP objective values (LP^{I} and IP^{I}) as well as the *deflated* LP and IP objective values (LP^{D} and IP^{D}). All four statistics are calculated on one run of the decomposed model. The inflated LP and IP values are simply the sum of the respective LP and IP objective values found in each of the blocks. This is an inaccurate indication of the total cost since the cost contribution from each block overlap is counted twice. IP^{D} is the true cost of the 63-day roster. This value is obtained by correctly adjusting the costs incurred in the overlaps. LP^{D}, on the other hand, is obtained by reducing the LP^{I} by the difference between the IP^{I} and IP^{D}. Owing to the heuristic shift fixing in the overlaps one cannot obtain an optimal decomposed LP solution. The purpose of the LP^{D} is to provide a lower bound when the non-decomposed bound (LP^{N}) cannot be obtained (ie for longer time horizons). Table 2 also reports the time taken to solve the decomposed model (t^{D}_{IP}) as well as the time taken to solve the LP for the non-decomposed model (t^{N}_{LP}), both of which are in seconds. The percentage gap between the deflated LP and IP objective values (G^{D}) and the optimality gap between the decomposed IP objective value and the non-decomposed LP objective values (G^{N}) are also given.

One can observe that the algorithm performs very efficiently with small optimality gaps between the best found integer solution using the decomposition approach and the optimal LP objective value for the non-decomposed model. Excluding instances A02, A03, and A09, all optimality gaps are less than 0.20%. The fact that $LP^{D} \approx LP^{N}$ indicates that the larger part of the gap is the integrality gap, caused by the relaxation of the integrality property for x_r. Hence, we conclude that the block structure does not reduce solution quality significantly. It is very encouraging to see that the block decomposition produces close to optimal integer solutions much faster than it can find the optimal LP objective value of the non-decomposed model. Furthermore, LP^{D} appears to be a slightly pessimistic approximation of LP^{N} as it is smaller in all but one instance (A02). However, both gaps, in general, are small enough that one can be confident of the superior performance of the algorithm. Finally, the results suggest that the real-life instances can be solved extremely efficiently, while it is just the artificial instances that create some difficulties.

A much more dramatic comparison is given in Table 3, where IP is compared with the best found integer solution to the non-decomposed model IPN. For the latter, a maximum of 150 000 s of computing time is permitted. Table 3 shows that there is very little difference in solution quality using the two approaches; however, the decomposition approach requires orders of magnitude less computing time. The fact that IP^{D} is better than IPN on some instances is simply due to the heuristic nature of the branching routine.

Table 4 gives the results of similar experiments in which the rostering horizon is increased from 63 days to 189 days. The results are very similar to those of Table 2, with a

small percentage gap between the IP^D and the LP^D as well as acceptable running times, particularly for the real-life instances. One can also observe that the best found solutions are approximately a factor three more than those found for the 63-day experiments ($3IP^D_{63}$). As was mentioned earlier, there is no guarantee that a solution with the latter objective value is even feasible. In some cases, that is A02, A05, A09, S10, and W07, the value IP^D is at least as good as ($3IP^D_{63}$). This is due to the unpredictable behaviour of the heuristic shift fixing and the branching routines in the block decomposition.

A particularly interesting graph is given in Figure 5 which illustrates the cost incurred on each day of the rostering horizon for S10. It is clear that the cost has a cyclic behaviour. A closer inspection shows that the costs are highly dependent on the weekday, which is not surprising, as the workload estimation is by weekday, but not over weeks. The workload graph for a day of the horizon with highest cost, Day 12, is shown in Figure 6.

Table 3 Decomposition *versus* no decomposition—63 days

Instance	LP^N	IP^D	$t^D_{IP}(s)$	IP^N	$t^N_{LP}(s)$
A01	39.90	39.90	110.67	39.90	13 545.69
A02	250.36	254.14	805.84	253.08	132 323.42
A03	242.17	243.31	692.48	243.56	139 643.58
A04	495.97	496.12	236.87	496.08	37 572.89
A05	218.57	218.82	1122.08	*	>150 000.00
A06	4799.94	4800.59	711.48	4800.73	89 770.27
A07	0.35	0.35	27.29	0.35	1273.07
A08	1099.60	1099.69	130.69	1099.72	13 325.80
A09	152.07	155.86	2079.08	*	>150 000.00
A10	3172.17	3173.90	713.56	3174.10	47 011.49
S08	0.00	0.00	15.68	0.13	124.17
S10	217.48	217.61	131.89	217.60	15 617.63
W07	0.57	0.57	39.45	0.57	3794.31

Day 12 is a Friday and all the larger costs of the horizon are observed on Fridays. It is clear that even on the worst days, the cover is fairly good and the major contribution to the cost is from the robustness measure. S10 is the realistic instance, where it is, by far, the most difficult to find a satisfactory cover. As the figure illustrates, even the worst day here has an acceptable cover. We conclude that the proposed method is very well suited for solving the problems at hand.

As can be seen from Table 4, the worst cover is obtained in instance A06. To sketch the worst case scenario, Figure 7 shows the workload graph of the most costly day of A06. Indeed, the amount of uncovered work is severe, but it is observed that this not due to a bad distribution of available manpower. The assigned manpower covers the estimated workload tightly, but there are simply too few employees to cover all the work. Also, this particular instance has a very jagged workload estimation, which makes it difficult to cover the highest peeks. The artificial instances were introduced to stress test the algorithm and to show what happens, when very hard instances are encountered. The figure illustrates that A06 is well suited for this purpose.

Table 5 provides a breakdown of the total cost (IP) into total roster-line costs (RC) and total coverage costs (CC). The RC component indicates how many late to early sequences are not satisfied in the chosen roster-lines, while CC states the cost incurred from uncovered workload and uncovered robustness. In Table 5, we also give both the number of uncovered workload hour on average per week (UWL) and the number of uncovered hours of robustness on average per week (UR). Here one can see that uncovered hours of robustness is the main component of the coverage cost in most cases. It is not surprising to see that the instances with the largest number of uncovered workload hours (A06, A08, and A10) also have the largest RC cost contribution. Here the model is simply attempting

Table 4 Results for a 189-day rostering horizon

Instance	LP^I	IP^I	LP^D	IP^D	$3IP^D_{63}$	G^D (%)	t^D_{IP} (s)
A01	198.76	198.76	120.19	120.19	119.70	0.00	249.19
A02	1265.19	1274.04	751.73	760.58	762.42	1.18	1581.17
A03	1213.80	1220.55	724.89	731.64	729.93	0.93	1040.34
A04	2485.13	2485.69	1488.10	1488.66	1488.36	0.04	555.58
A05	1096.74	1098.43	656.11	657.80	656.46	0.26	1920.91
A06	23 936.35	23 941.99	14 405.78	14 411.42	14 401.77	0.04	798.84
A07	1.78	1.78	1.05	1.05	1.05	0.00	65.30
A08	5470.04	5470.24	3310.39	3310.59	3299.07	0.01	251.30
A09	768.96	782.49	453.08	466.60	467.58	2.98	2753.54
A10	15 686.69	15 695.55	9520.05	9528.92	9521.70	0.09	945.74
S08	3.30	3.30	3.30	3.30	0.00	0.00	54.32
S10	1047.70	1047.26	648.05	648.61	652.83	0.09	322.95
W07	2.87	2.87	1.70	1.70	1.71	0.00	121.58

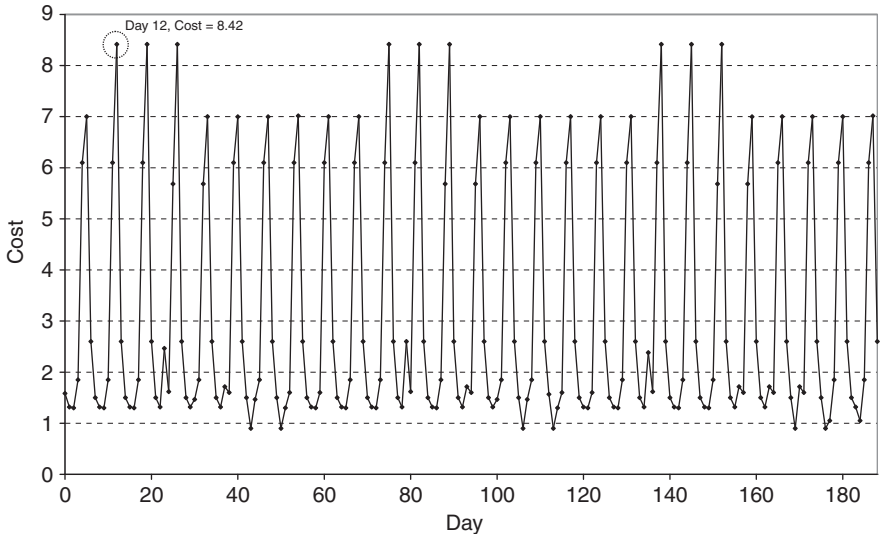

Figure 5 The cost incurred on each day of the rostering horizon for S10.

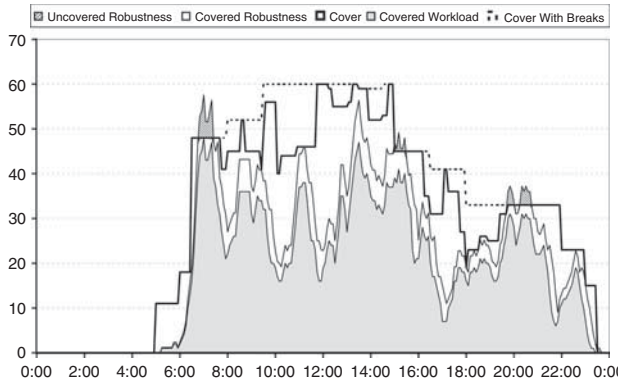

Figure 6 Workload graph for the most costly day of S10 with the cover of the found solution.

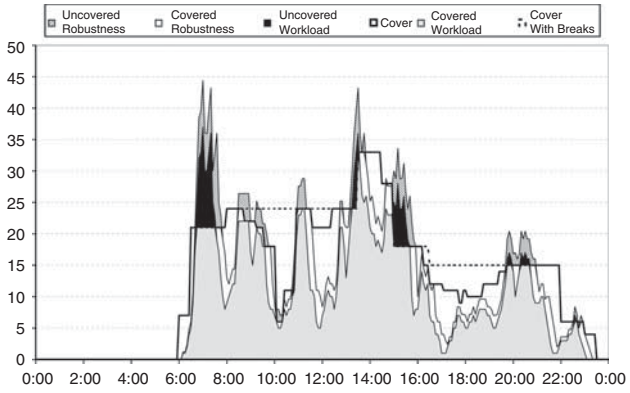

Figure 7 Workload graph for the most costly day of A06 with the cover of the found solution.

to cover the workload as well as possible, often disregarding the late to early sequence preference for the roster-line. Finally, IE gives the *Implied Efficiency* of the obtained roster. Implied efficiency is the percentage of time that people at work are actually working.

Table 5 Solution statistics for 189-day rosters

Instance	*IP*	*RC*	*CC*	*UWL*	*UR*	*IE (%)*
A01	120.19	0.00	120.19	0.00	4.45	59.28
A02	760.58	17.00	743.58	0.53	22.24	65.12
A03	731.64	0.00	731.64	0.10	26.10	61.52
A04	1488.66	6.00	1482.66	1.23	42.61	62.11
A05	657.80	0.00	657.80	0.02	24.16	65.18
A06	14 411.42	593.00	13 818.43	37.92	132.59	59.08
A07	1.05	0.00	1.05	0.00	0.04	50.53
A08	3310.59	373.00	2937.59	6.60	42.80	56.90
A09	466.60	0.00	466.60	0.00	17.28	67.77
A10	9528.92	916.00	8612.92	18.73	131.70	62.12
S08	3.30	2.00	1.30	0.00	0.05	45.98
S10	648.61	60.00	588.61	0.03	21.50	55.30
W07	1.70	0.00	1.70	0.00	0.06	57.07

To provide an indication of how the solutions to S08, S10, and W07 in Table 4 compare to the airline's solutions, Table 6 makes a comparison of the average number of uncovered workload hours per week, the average number of uncovered hours of robustness, and the implied efficiency of each of the rosters. In Table 6, column headings with an A superscript denote the airline's value. For all instances, we perform much better, significantly improving the robustness of the roster. It is not possible to improve implied efficiency without reducing the staffing level. Instances S08′, S10′, and W07′ are identical to S08, S10, and W07 in which the staffing level has been reduced by 10–12%. Here we see that we can provide a similar coverage and more robustness than the airline, while at the same time improving efficiency by around 7–11%.

Table 6 Comparison with airline's solutions

Instance	UWL^A	*UWL*	UR^A	*UR*	*RC*	IE^A	*IE*
S08	4.42	0.00	12.52	0.05	0.00	46.10	45.98
S10	0.00	0.03	147.69	21.50	60.00	55.29	55.30
W07	0.00	0.00	41.28	0.06	0.00	57.14	57.07
S08′	4.42	0.00	12.52	0.01	0.00	46.10	51.52
S10′	0.00	0.33	147.69	36.40	63.00	55.29	59.57
W07′	0.00	0.00	41.28	2.56	0.00	57.14	61.60

Although instance S10 has 0.33 uncovered workload hours per week on average, this equates to around 20 min per week and can be considered negligible. It should also be mentioned that in increasing the efficiency there is only a very slight increase in the RC for instance S10.

9. Conclusion

In this paper, we have considered the GCRPWP arising at a major European airline. We have proposed a cutting stock-based integer programming formulation of the problem, which is not only able to circumvent one step of the roster construction process but which can also accurately incorporate the necessary robustness measures. A powerful decomposition approach utilizing column generation and variable fixing is developed to solve a sequence of integrated optimization problems. This is combined with a shift fixing routine to ensure the roster-lines obtained for each of the smaller problems can be pieced together to construct a roster for the entire six month planning horizon. Computational results on three real-life instances and 10 artificial instances confirm the efficiency of the proposed methodology. Not only do we find better solutions than those implemented by the airline, particularly from a robustness perspective, but we have also shown that more robust solutions can be obtained even if staffing levels are reduced by 10–12%.

References

Abbink E, Fischetti M, Kroon L, Timmer G and Vromans M (2005). Reinventing crew scheduling at Netherlands railways. *Interfaces* **35**(5): 393–401.

Alfares HK (1997). An efficient two-phase algorithm for cyclic days-off scheduling. *Comput Opns Res* **25**(11): 913–923.

Alfares HK (2002). Optimum workforce scheduling under the (14, 21) days-off timetable. *J Appl Math Decis Sci* **63**(3): 191–199.

Amor HB and de Carvalho JV (2005). Cutting stock problems. In: Desaulniers G, Desrosiers J and Solomon M (eds). *Column Generation*. Chapter 5. Springer: New York, pp 131–161.

Anbil R, Gelman E, Patty B and Tanga R (1991). Recent advances in crew-pairing optimization at American Airlines. *Interfaces* **21**(1): 62–74.

Barnhart C, Cohn AM, Johnson EL, Klabjan D, Nemhauser GL and Vance PH (2003). Airline crew scheduling. In: Hall RW (ed). *Handbook of Transportation Science*. Kluwer Academic Publishers: Dordrecht.

Brusco MJ, Jacobs LW, Bongiorno RJ, Lyons DV and Tang B (1995). Improving personnel scheduling at airline stations. *Ops Res* **43**(5): 741–751.

Burke EK, De Causmaecker P, vanden Berghe G and van Landeghem H (2004). The state of the art of nurse rostering. *J Scheduling* **7**: 441–499.

Burke EK, De Causmaecker P, De Maere G, Mulder J, Paelinck M and Vanden Berghe G (2010). A multi-objective approach for robust airline scheduling. *Comput Opns Res* **37**(5): 822–832.

Butchers ER, Day PR, Goldie AP, Miller S, Meyer JA, Ryan DM, Scott AC and Wallace CA (2001). Optimized crew scheduling at Air New Zealand. *Interfaces* **31**(1): 30–56.

Cheang B, Li H, Lim A and Rodrigues B (2003). Nurse rostering problems—A bibliographic survey. *Eur J Opl Res* **151**: 447–460.

Chu SCK (2007). Generating, scheduling and rostering of shift crew-duties: Applications at the Hong Kong International Airport. *Eur J Opl Res* **177**: 1764–1778.

Clausen J, Larsen A, Larsen J and Rezanova NJ (2010). Disruption management in the airline industry – Concepts, models and methods. *Comput Opns Res* **37**(5): 809–821.

Danna E and Pape CL (2005). Branch-and-price heuristics: A case study on the vehicle routing problem with time windows. In: Guy Desaulniers JD and Solomon MM (eds) *Column Generation*. Chapter 4. Springer: New York, pp 99–129.

Day PR and Ryan DM (1997). Flight attendant rostering for short-haul airline operations. *Opns Res* **45**(5): 649–661.

Desaulniers G (2010). Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. *Opns Res* **58**(1): 179–192.

Desaulniers G, Desrosiers J, Ioachim I, Solomon M, Soumis F and Villeneuve D (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In: Crainic T and Laporte G (eds). *Fleet Management and Logistics*. Chapter 3. Kluwer Academic Publishers: Dordrecht, pp 57–94.

Desaulniers G, Desrosiers J and Solomon MM (2002). *Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems*. Kluwer: Norwell, pp 309–324.

Desrochers M and Soumis F (1988). A reoptimization algorithm for the shortest path problem with time windows. *Eur J Opl Res* **35**(2): 242–254.

Desrosiers J and Lübbecke ME (2005). Column generation. In: Desaulniers G, Desrosiers J and Solomon MM (eds). *A Primer in Column Generation*. Chapter 1. Springer: New York, pp 1–32.

Dohn A, Mason A and Ryan D (2010). *A generic solution approach to nurse rostering*. Technical Report 5, Department of Management Engineering, Technical University of Denmark.

Dowling D, Krishnamoorthy M, Mackenzie H and Sier D (1997). Staff rostering at a large international airport. *Ann Opns Res* **72**: 125–147.

Ernst A, Jiang H, Krishnamoorthy M and Sier D (2004). Staff scheduling and rostering: A review of applications, methods, and models. *Eur J Opns Res* **153**: 3–27.

Eveborn P and Rönnqvist M (2004). Scheduler—A system for staff planning. *Ann Opns Res* **128**(1–4): 21–45.

Gendreau M, Ferland J, Gendron B, Hail N, Jaumard B, Lapierre S, Pesant G and Soriano P (2006). Physician scheduling in emergency rooms. In: Burke EK and Rudová H (eds). *Practice and Theory of Automated Timetabling VI. 6th International Conference, PATAT*. Springer: Verlag.

Irnich S (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spect* **30**(1): 113–148.

Irnich S and Desaulniers G (2005). Shortest path problems with resource constraints. In: Desaulniers G, Desrosiers J, Solomon M (eds). *Column Generation*. Chapter 2. Springer: New York, pp 33–66.

Kellogg DL and Walczak S (2007). Nurse scheduling: From academia to implementation or not? *Interfaces* **37**(4): 355–369.

Mehrotra A, Murphy K and Trick M (2000). Optimal shift scheduling: A branch-and-price approach. *Naval Res Logist* **47**(3): 185–200.

Ryan D and Foster B (1981). An integer programming approach to scheduling. In: Wren A (ed). *Computer Scheduling of Public Transport. Urban Passenger Vehicle and Crew Scheduling*. North-Holland, Amsterdam, pp 269–280.

Vohra RV (1987). The cost of consecutivity in the (5, 7) cyclic staffing problem. *IIE Trans* **29**: 942–950.

Wäscher G and Gau T (1996). Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spect* **18**(3): 131–144.

Weide O, Ryan D and Ehrgott M (2010). An iterative approach to robust and integrated aircraft routing and crew scheduling. *Comput Opns Res* **37**(5): 833–844.

Received May 2010;
accepted February 2011 after one revision